Vehicle-to Everything Communication Project

This work is supported by NSF grants 2150292 and 2150096
Overview

- Motivation
- Background
- Objectives
- Experimental Setup
- Methods
- Future Work
Goals of V2X
Improve Safety

- Average human reaction time: ¾ of a second [1]
- Nearly instant communication
- Identify and communicate hazards, pedestrians, poor weather conditions, and more

Move Efficiently

• Optimize intersections

• Harmonize speed to allow vehicles to follow each other closely

• Reduced fuel emissions with platooning and efficient intersections [2]

Current State of V2X
Vehicle Ad-Hoc Network

- A VANET provides internet access and connectivity between vehicles [3]

- VANET’s are being researched to prevent collisions, dynamically schedule routes, and monitor conditions [3]

Vehicle Infrastructure Integration

- Siren sensors
- Road-side units
- Weather sensors
- Traffic detection
- License plate recognition
Primary Objectives

1. Create connections between vehicles
2. Map real-time vehicle locations via a Web GUI
3. Create occupancy grids
Experimental Setup

HARDWARE

Roadside Unit

Autonomous Campus Transport (ACTor) Vehicle
Time, Position, Navigation

SOFTWARE STACK

C++

ROS

Python

MariaDB

NodeJS & HTML

SocketIO

Google Maps API
Swift Navigation

- Swift Piksi Multi Receivers
  - GNSS with GPS and RTK capabilities
  - ECEF, UTM, LTP, LLH formats
  - Reference and Attitude Receivers
  - ROS compatible
Time, Position, Navigation

- `tpn_node`
- Subscribe to LLH and DBW topic
- Custom Position Message
  - Latitude, Longitude, Heading, Velocity, and Angular Z
- Future location prediction
- Live map updates
Object Detection

SOFTWARE STACK

- C++
- ROS
- Python
- MariaDB
- NodeJS & HTML
- SocketIO
- Google Maps API
Database

SOFTWARE STACK

C++
ROS
Python
MariaDB
NodeJS & HTML
SocketIO
Google Maps API
## Client Vehicle Table

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Unique identifier label for each vehicle connected to the RSU</td>
<td>INT</td>
</tr>
<tr>
<td>latitude</td>
<td>Most recent latitude of the vehicle</td>
<td>FLOAT</td>
</tr>
<tr>
<td>longitude</td>
<td>Most recent longitude of the vehicle</td>
<td>FLOAT</td>
</tr>
<tr>
<td>heading</td>
<td>Most recent heading of the vehicle</td>
<td>FLOAT</td>
</tr>
<tr>
<td>speed</td>
<td>Most recent linear x velocity of the vehicle</td>
<td>FLOAT</td>
</tr>
<tr>
<td>angular_z</td>
<td>Most recent angular z velocity of the vehicle</td>
<td>FLOAT</td>
</tr>
</tbody>
</table>
# Identified Objects Table

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Unique identifier label for each object detected</td>
<td>INT</td>
</tr>
<tr>
<td>latitude</td>
<td>Most recent latitude of the object</td>
<td>FLOAT</td>
</tr>
<tr>
<td>longitude</td>
<td>Most recent longitude of the object</td>
<td>FLOAT</td>
</tr>
<tr>
<td>type</td>
<td>Type of object detected</td>
<td>CHAR(50)</td>
</tr>
<tr>
<td>time</td>
<td>Time that the object was detected</td>
<td>CHAR(8)</td>
</tr>
</tbody>
</table>
Remote Host

Any computer can connect to the database!
### Client Vehicle Table

<table>
<thead>
<tr>
<th>index</th>
<th>latitude</th>
<th>longitude</th>
<th>heading</th>
<th>speed</th>
<th>angular_z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>37.7749</td>
<td>-122.491</td>
<td>90.5</td>
<td>50.2</td>
<td>1.8</td>
</tr>
<tr>
<td>2</td>
<td>34.0522</td>
<td>-118.244</td>
<td>180.0</td>
<td>45.8</td>
<td>-0.5</td>
</tr>
</tbody>
</table>

### Identified Objects Table

<table>
<thead>
<tr>
<th>index</th>
<th>latitude</th>
<th>longitude</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>37.1234</td>
<td>-122.568</td>
<td>Cone</td>
</tr>
<tr>
<td>2</td>
<td>38.4321</td>
<td>-121.877</td>
<td>Cone</td>
</tr>
<tr>
<td>3</td>
<td>39.5678</td>
<td>-120.988</td>
<td>Pedestrian</td>
</tr>
</tbody>
</table>
**Web GUI**

**SOFTWARE STACK**

<table>
<thead>
<tr>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROS</td>
</tr>
<tr>
<td>Python</td>
</tr>
<tr>
<td>MariaDB</td>
</tr>
<tr>
<td>NodeJS &amp; HTML</td>
</tr>
<tr>
<td>SocketIO</td>
</tr>
<tr>
<td>Google Maps API</td>
</tr>
</tbody>
</table>
Web GUI - Development

- NodeJS, GoogleMaps API, & MariaDB → plot and display markers per query
- Pull from the database every 500 ms - SocketIO
- Verify represented objects
Blue Marker: ACTor 1
Green Marker: ACTor 2
Orange Marker: Object
Web GUI - Features

Automatically updates based on info from DB
Occupancy Grid

SOFTWARE STACK

C++
ROS
Python
MariaDB
NodeJS & HTML
SocketIO
Google Maps API
Occupancy Grid

- Manage intersections
- Plot vehicles connected to the RSU
- Predict their paths
- Plot obstructions detected by other vehicles
- Detect and avoid collisions
Tracking Vehicles:
actor1
actor2

Events:
actor1 hits actor2
Time to Impact: 1.51 s.
Actions: stop actor1
actor1 hits actor2
Time to Impact: 1.85 s.
Actions: stop actor1
actor1 hits actor2
Time to Impact: 1.97 s.
Actions: stop actor1
Trajectory Prediction

- Vehicles are stored with a unique ID
- Velocity and steering angle can be used to predict paths
- Collisions are avoided by changing velocity
Remote Speed Limit Control

SOFTWARE STACK

C++
ROS
Python
MariaDB
NodeJS & HTML
SocketIO
Google Maps API
Remote Speed Limit Control

- Config and broadcast speed limit remotely
  - Rosnode on Raspberry Pi
  - Command line parameter on launch

- Enforce speed limit with firewall node
  - Receive speed limit from Raspberry Pi
  - Limit speed to vehicle
Remote Speed Limit Control
Future Work for Our Project

- Finish integration in vehicles
- Expand network to 2+ roadside units
- Expand object detection to more and unfamiliar obstacles
- Time relevance of objects based on position
Future Work for V2X

1. Reduce traffic on highways

Future Work for V2X

2. Eliminate need for turn signals and traffic lights [4]

Future Work for V2X

3. Implement path scheduling with occupancy grid [5]
References


Thank you!
Questions?
Let’s see it in action!
Let’s head out to the track!